eurasien (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), brevet européen (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), brevet OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Publiée :

— *sans rapport de recherche internationale, sera republiée dès réception de ce rapport*

En ce qui concerne les codes à deux lettres et autres abréviations, se référer aux "Notes explicatives relatives aux codes et abréviations" figurant au début de chaque numéro ordinaire de la Gazette du PCT.

Sécurisation d'application téléchargée notamment dans une carte à puce

La présente invention concerne la sécurisation de l'environnement d'exécution d'un interpréteur, tel qu'une machine virtuelle, dans un dispositif de traitement de données du type objet électronique portable telle qu'une carte à puce.

Plus particulièrement l'invention concerne la protection contre des attaques pour exécuter des données comme si elles étaient des instructions (opcodes). Une attaque classique consiste par exemple à exécuter un saut indésiré vers une zone de mémoire qui a enregistré des données. Ceci résulte par exemple de la modification de pseudo-codes dans une application téléchargée, telle qu'une applet, par un attaquant afin d'introduire une applet "agressive" dans un champ de données et ainsi introduire un saut vers cette partie de données. La machine virtuelle, qui n'est pas capable de distinguer les instructions et des données, exécute alors les données comme si elles étaient des instructions.

L'invention vise à protéger le fonctionnement de la machine virtuelle contre de telles attaques et plus précisément à distinguer les instructions et les données particulièrement lors de l'exécution d'une partie d'une application téléchargée.

Un dispositif de traitement de données selon l'invention comprend un moyen de mémorisation pour mémoriser au moins une application téléchargée, initialement compilée en un langage intermédiaire, composée de plusieurs composants applicatifs

contenant chacun un identificateur et des mots d'instructions, et un moyen d'exécution virtuel. Il est caractérisé en ce qu'il comprend :

- un moyen générateur de nombres aléatoires pour associer un nombre aléatoire à un composant applicatif prédéterminé de l'application téléchargée,

- un premier moyen transformateur inclus dans le moyen d'exécution virtuel pour appliquer chacun des mots d'instruction dans le composant applicatif prédéterminé et le nombre aléatoire associé à une fonction de transformation afin de mémoriser des mots d'instruction transformés lors du téléchargement du composant applicatif prédéterminé, et

- un deuxième moyen transformateur inclus dans le moyen d'exécution virtuel pour appliquer chacun des mots d'instruction transformés d'une partie du composant applicatif prédéterminé et le nombre aléatoire associé à la fonction réciproque de la fonction de transformation afin de récupérer les mots d'instruction composant ladite partie du composant applicatif prédéterminé pour exécuter ladite partie de composant ainsi récupérée.

Par exemple, le composant applicatif prédéterminé comporte une suite de méthodes, en tant que partie du composant, dont les mots d'instruction, tels que des octets de code opération et des octets de paramètre, subissent systématiquement la transformation en mots d'instruction transformés avant d'être enregistrés de manière permanente dans le dispositif de traitement de données, ce qui permet de distinguer les instructions des données. Par exemple, un saut introduit irrégulièrement dans une application téléchargée ne permettra pas d'aboutir à une donnée recherchée puisque l'exécution du saut se

fera par rapport à une donnée réputée transformée qui ne correspond pas au saut recherché.

Afin d'accentuer encore la distinction entre les instructions de code et les données, le moyen générateur génère un nombre aléatoire produit par le moyen d'exécution virtuelle, et le premier moyen transformateur applique, par exemple lors de la création d'un objet par le moyen d'exécution virtuel, chaque donnée et le nombre aléatoire associé à la fonction de transformation afin d'écrire une donnée transformée dans le moyen de mémorisation lorsque la donnée est produite par le moyen d'exécution virtuel. En variante, l'invention prévoit de distinguer les différentes données en fonction des types primitifs des données. Dans cette variante, le moyen générateur génère des nombres aléatoires respectivement associés à des types primitifs de données, et le premier moyen transformateur applique chaque donnée produite par le moyen d'exécution virtuel et le nombre aléatoire associé au type primitif de la donnée à la fonction de transformation afin d'écrire notre donnée transformée dans le moyen de mémorisation lorsque la donnée est produite par le moyen d'exécution virtuel.

D'autres caractéristiques et avantages de la présente invention apparaîtront plus clairement à la lecture de la description suivante de plusieurs réalisations préférées de l'invention en référence aux dessins annexés correspondants dans lesquels :

- la figure 1 est un bloc-diagramme schématique d'un dispositif de traitement de données du type carte à puce selon l'invention ;

- la figure 2 est un algorithme de chargement d'une application dans le dispositif de traitement de données selon l'invention ;

- la figure 3 est un diagramme montrant une suite d'instructions notamment lors de la création d'un objet ;

- la figure 4 est un algorithme de création d'objet selon les instructions de la figure 3 ;

- la figure 5 est un diagramme d'instructions transformées dans une méthode pour exécuter une addition de deux variables locales selon l'application chargée ; et

- la figure 6 est un algorithme d'exécution de la méthode avec les instructions transformées montrée à la figure 5.

Dans la suite de la description, on se réfèrera à une carte à puce 1, dite également carte à microcontrôleur ou à circuit intégré, en tant qu'objet électronique portable logé d'une manière amovible dans un lecteur 21 d'une plate-forme d'accueil telle qu'un terminal d'accueil 2. La carte à puce 1 est de n'importe quel type connu de carte à puce à contact ou sans contact, et peut être une carte de paiement, une carte téléphonique, une carte additionnelle, un module d'identité d'abonné téléphonique amovible SIM (Subscriber Identity Module), une carte de jeu, etc…

Le terminal d'accueil 2 peut être un ordinateur personnel PC ou un terminal bancaire ou un terminal point de vente, ou bien encore un terminal radiotéléphonique cellulaire mobile, ou un objet électronique portable tel qu'un assistant numérique personnel PDA (Personal Digital Assistant) ou un porte-monnaie électronique.

Le microcontrôleur dans la carte à puce 1, en tant que dispositif de traitement de données, comprend un microprocesseur 3, une mémoire non

réinscriptible 4 du type ROM, une mémoire non volatile 5 de type EEPROM et une mémoire à accès aléatoire 6 de type RAM. Tous les composants 3 à 6 dans la carte à puce 1 sont reliés par un bus 7 interne à la carte et une interface de communication reliée au lecteur 21 dans le terminal d'accueil 2 à travers une liaison à contact ou sans contact LI.

Des espaces 40 et 41 de la mémoire 4 contiennent respectivement des instructions en code natif d'un système d'exploitation OS (Operating System) et en pseudo-code (bytecode) d'une machine virtuelle VM, en tant que moyen d'exécution virtuelle, sur laquelle s'appuie le système d'exploitation. Les pseudo-codes résultent de la compilation d'un programme en langage source de haut niveau du type orienté objet, tel que par exemple le langage Java Card. Par exemple un serveur (non représenté) comprenant un compilateur convertit le programme en langage source Java Card en un programme compilé en langage intermédiaire, c'est-à-dire en pseudo-codes qui sont des mots d'instruction formés par des octets, appelés "bytecodes", qui sont prêts à être exécutés par la machine virtuelle VM, en tant qu'interpréteur dans la carte à puce 1. Le programme compilé constitue une application AP, dite applet, téléchargée dans la carte à puce au sens de l'invention. La mémoire 4 comprend également au moins des applications d'authentification et de communication internes à la carte.

Des espaces 50 et 51 de la mémoire non volatile 5 contiennent respectivement des données liées au système d'exploitation OS et accessibles par codes natifs et des données liées à la machine virtuelle VM et accessibles par pseudo-codes, ainsi que les pseudo-codes et les données d'applications

téléchargées dans la carte. La mémoire 5 contient également des données personnelles liées au possesseur de la carte à puce.

La mémoire 6 de type RAM contient essentiellement des données échangées avec le monde extérieur à la carte à puce 1, notamment avec le terminal d'accueil TE.

La mémoire 6 comprend notamment un espace prédéterminé de taille fixe pour recevoir une ou des applications téléchargées AP, telles que des applets, ou des portions d'application, depuis un serveur à travers le terminal d'accueil 2 et la liaison LI pour être exécutées par la machine virtuelle VM. L'espace prédéterminé est divisé en trois espaces de mémoire 60, 61 et 62.

L'espace 60 sert principalement de mémoire tampon pour recevoir des données de machine, telles qu'une application téléchargée AP et la transformer selon l'invention en une application transformée qui est écrite dans l'espace de mémoire 51 de la mémoire 5 réservée aux données de la machine virtuelle VM.

Les deux autres espaces de mémoire 61 et 62 sont réservés à des premières parties de méthodes invoquées dans des applications comprenant des variables locales VL et à des deuxièmes parties ayant des tailles variables et comprenant des opérandes OP des méthodes invoquées. L'invocation d'une méthode sur le dessus de la pile ainsi constituée dans la mémoire 6 sous le contrôle du processeur 3 provoque l'empilement d'un cadre ("frame") respectif sur le dessus de la pile qui contient les autres cadres de méthode. Les méthodes s'invoquent les unes les autres, la méthode précédente invoquant la méthode suivante et la méthode suivante ne pouvant que retourner à la méthode précédente, en dépilant et

écartant la méthode du dessus de la pile. Ainsi seulement la méthode au-dessus de la pile est active.

La largeur de la pile est par exemple égale à un octet, soit égale à la longueur d'un pseudo-code.

Selon la technique antérieure, une ou plusieurs variables locales VL dans l'espace de mémoire 61 sont déclarées lors de l'implémentation d'une méthode et avant l'exécution de celle-ci. Les variables locales servent à l'exécution de la méthode, leur nombre n'étant pas modifié mais leurs valeurs pouvant être modifiées au cours de l'exécution de la méthode. Comme on le verra dans la suite, à titre d'exemple, une variable locale peut être une référence à un objet dont la méthode est appelée afin d'accéder à cet objet dans la machine virtuelle VM, ou bien des paramètres de la méthode ou d'autres variables locales.

Egalement selon la technique antérieure, des opérandes dans l'espace de mémoire 62 sont des valeurs utilisées par la machine virtuelle VM pour exécuter des prochaines opérations et en particulier utilisées comme argument de l'appel de la méthode invoquée. Des opérandes d'une méthode peuvent devenir des variables locales de la méthode suivante lors de l'implémentation de la méthode suivante, et inversement le résultat d'une méthode peut devenir un opérande de la méthode immédiatement précédente lors du retour à celle-ci.

Une application (applet) est sous la forme de programmes compilés et structurés en plusieurs composants applicatifs logiciels CP comprenant chacun une suite d'octets (bytecodes).

Chaque octet supporte un code opération (opcode) constituant une instruction IN proprement dite, ou

bien l'un des paramètres PA d'une instruction appelés opérandes. Une instruction proprement dite est ainsi composée d'un octet d'instruction IN (opcode) qui est éventuellement suivi d'un ou de plusieurs octets de paramètre PA.

L'application à télécharger à laquelle on se réfèrera dans la suite à titre d'exemple est sous la forme d'un fichier compilé du type cap.file et comporte notamment un composant de méthode CP1 qui contient toutes les méthodes de l'application à télécharger AP. Chaque méthode a une longueur fixe et comprend plusieurs groupes consécutifs ayant chacun un octet d'instruction IN et un nombre prédéterminé d'octets de paramètre PA. Un autre composant CP2 de l'application AP peut être un composant de champ statique qui contient tous les champs statiques des classes de l'application. Un champ statique est un champ qui appartient à une classe indépendamment de toute instance éventuelle et n'est pas une instance d'un objet de la classe et est ainsi partagée par toutes les instances de la classe. Un champ statique est comme un attribut associé à tous les objets d'une classe.

Comme montré également à la figure 1, la carte à puce 1 comprend en outre, selon l'invention, un générateur de nombres aléatoires 30 et deux transformateurs logiques 42 et 43.

Selon la réalisation illustrée, le générateur 30 est implémentée matériellement dans ou en liaison avec le microprocesseur 3 de la carte à puce. Il échange des requêtes et des réponses à travers le bus 7 avec la machine virtuelle VM dans l'espace de mémoire 40 afin de générer des nombres aléatoires NA. Par exemple, le générateur 30 associe un nombre

aléatoire à un composant applicatif prédéterminé dans l'application AP lors du chargement de celle-ci, ou bien à des données notamment pour la création d'objet, comme on le verra dans la suite.

Selon une autre réalisation, le générateur de nombres aléatoires est inclus sous forme de logiciel dans la machine virtuelle VM, c'est-à-dire dans l'espace de mémoire 40.

Chaque fois que le générateur de nombres aléatoires 30 génère un nombre aléatoire NA, le nombre aléatoire généré NA est écrit dans un registre RG inclus dans l'espace de données de machine 60 dans la mémoire RAM 6, sous la commande de la machine virtuelle VM. La machine virtuelle fait correspondre la valeur du pointeur dans la mémoire 6 au niveau du registre RG à un identificateur ID inclus dans un en-tête du composant CP auquel est associé le nombre aléatoire NA. La correspondance entre l'identificateur ID et le registre RG est écrite dans l'espace de mémoire 51 alloué à la machine virtuelle VM dans la mémoire EEPROM 5.

Les transformateurs 42 et 43 sont implémentés préalablement sous forme logiciels dans l'espace de mémoire 41 de la mémoire ROM 4 et sont ainsi inclus dans la machine virtuelle VM.

Le premier transformateur 42 transforme un composant applicatif prédéterminé CP1 en un composant transformé CPT1 = FT(CP1, NA1) résultant de l'application du composant prédéterminé CP1 et d'un nombre aléatoire NA1 qui lui est associé à une fonction de transformation FT. Une telle transformation est effectuée par exemple octet par octet sur tous les octets OC1 du composant prédéterminé CP1 lors du chargement de celui-ci dans l'espace de mémoire de données de machine 60 dans la

mémoire RAM 6 ou sur chaque opérande lors de l'écriture d'un opérande, tel qu'une référence d'objet REF, dans l'espace d'opérande 62 lors de la création d'un objet.

Inversement, lorsqu'une méthode doit être exécutée ou lorsqu'un opérande d'un objet doit être lu dans l'espace 62, une transformation FT^{-1} réciproque de la transformation FT et incluse dans le transformateur 43 transforme des octets transformés OCT1 dans une méthode de composant transformé, ou l'opérande transformé, tel que la référence d'objet transformée REFT, en les octets initiaux OC = FT^{-1}(OCT, NA1) du composant CP ou en la référence REF = FT^{-1}(REFT, NAD) par application des octets transformés OCT ou de la référence transformée REFT et du nombre aléatoire associé NA1, NAD à la fonction réciproque FT^{-1}.

Selon une réalisation préférée, la fonction de transformation FT est la fonction OU Exclusif (XOR) et par conséquent la fonction réciproque correspondante FT^{-1} est également la fonction OU Exclusif.

Selon d'autres variantes, bien d'autres fonctions de transformation logiques réversibles FT peuvent être choisies pour la mise en œuvre de l'invention. Par exemple, la fonction FT est la multiplication d'un octet et d'un nombre aléatoire et la fonction FT^{-1} est la division ; ou la fonction FT est l'addition d'un octet et d'un nombre aléatoire et la fonction FT^{-1} est la soustraction ; ou bien la fonction FT est un décalage d'un nombre de bits vers un côté, à droite ou à gauche, dans un octet OC, le nombre déterminant le décalage étant égal à un nombre aléatoire NA inférieur au nombre de bits de l'octet, et la fonction réciproque FT^{-1} est un décalage à

gauche ou à droite du nombre de bits NA dans l'octet transformé OCT.

Plus généralement, les transformateurs 42 et 43 peuvent traiter des mots d'instruction de longueur constante à plusieurs octets à transformer, au lieu simplement d'octets, par exemple correspondant à des instructions complètes.

En référence maintenant à la figure 2, le téléchargement d'une application AP dans la carte à puce 1 comprend essentiellement des étapes C1 à C8. Classiquement, l'application AP comportant plusieurs composants logiciels CP est écrite progressivement dans un registre RG1 de l'espace de mémoire de données de machine 60 dans la mémoire RAM 6. Les étapes suivantes C2 à C7 sont effectuées progressivement au fur et à mesure du téléchargement de l'application AP jusqu'au transfert d'un composant transformé de celle-ci de la mémoire RAM 6 vers la mémoire EEPROM 5 à l'étape C7.

Il est supposé que la transformation selon l'invention n'est appliquée qu'à au moins un composant prédéterminé CP1 dans l'application à télécharger AP désigné par un identificateur ID1 dans l'en-tête du composant CP1. La machine virtuelle VM détecte les identificateurs ID au début de chaque composant CP de l'application AP à l'étape C2 afin de déclencher, comme indiqué à l'étape C3, le générateur de nombres aléatoires 30 lorsque la machine virtuelle a détecté l'identificateur ID1 du composant CP1. A l'étape C4, le générateur 30 génère un nombre aléatoire NA1 qui est écrit dans un registre RG2 dans l'espace 60 de la mémoire ROM 6 et la machine virtuelle VM associe l'identificateur ID1 à une

valeur de pointeur liée au registre RG2 en écrivant cette correspondance dans l'espace 51.

A l'étape C5, pour chaque octet OC1 dans le composant CP1 en cours de chargement, la machine virtuelle VM écrit l'octet OC1 dans un registre tampon RG3 de l'espace de mémoire 60, le transformateur 42 applique l'octet OC1 lu dans le registre RG3 et le nombre aléatoire NA1 lu dans le registre RG2 à la fonction de transformation FT, telle que la fonction OU Exclusif, et écrit le résultat OCT1=FT(OC1,NA1) dans un registre RG4 de l'espace 60, et finalement la machine virtuelle VM remplace l'octet OC1 dans le registre RG1 par l'octet transformé correspondant OCT1 lu dans le registre RG4. Lorsque tous les octets OC1 ont été transformés en octets OCT1, le registre RG1 contient le composant transformé CPT1=FT(CP1,NA1), comme indiqué à l'étape C6. Le composant transformé CPT1 est ensuite transféré du registre RG1 dans l'espace 51 de la mémoire EEPROM 2.

Les étapes C2 à C7 sont répétées pour tout autre composant CP2 à transformer de l'application téléchargée AP. Le générateur 30 génère un nombre aléatoire NA2 qui l'associe au composant CP2 et l'écrit dans un autre registre dans l'espace de mémoire 60 afin de transformer chaque octet OC2 du composant CP2 en un octet transformé OCT2=FT(OC2,NA2). Par exemple le composant CP2 est un composant de champ statique qui contient des champs statiques des classes de l'application AP.

En variante, au lieu d'appliquer la fonction de transformation FT à un octet, la fonction de transformation est appliquée à un nombre prédéterminé d'octets. Par exemple dans le composant CP1 si toutes les instructions comprennent chacune un octet

d'instruction IN suivi de deux octets de paramètres PA, chaque mot M1=(IN,PA,PA) dans le composant CP1 est transformé en un mot transformé MT1=FT(M1,NA1) résultant de l'application du mot M1 et du nombre aléatoire NA1 associé au composant CP1 à la fonction de transformation FT.

Ainsi, la machine virtuelle VM charge le composant CP1, CP2 dans la carte à puce 1 en le masquant aléatoirement.

En variante, lorsque l'application ou un composant de celle-ci a une taille relativement grande et ne peut être entièrement chargé dans la mémoire RAM, l'application est découpée en portions de taille constante, comme des paquets. Les étapes de chargement C4 à C7 sont relatives à chaque portion de manière à transformer successivement les portions de l'application. Lorsqu'une portion constitue une transition entre deux composants concaténés, chaque partie de la portion à la fin ou au début de composant est reconnue et traitée séparément par les étapes C4 à C7. Chaque portion est chargée et transformée et finalement transférée dans l'espace 51 de la mémoire EEPROM 5, avant de télécharger la portion suivante dans la mémoire RAM 6.

Selon l'invention, la création d'un objet par exemple au cours de l'exécution d'instructions dans une application interne, comme montré à la figure 3, engendre la transformation de données liées à la création de l'objet, notamment la transformation au moins d'une référence REF à l'objet créé en une référence transformée REFT sans nécessiter la transformation réciproque de la référence transformée REFT.

Dans la figure 3, la première instruction "new" dans un octet de code opération (opcode) suivi de deux octets de paramètre "indexbyte" est d'abord exécutée pour créer un objet OB à une étape O2 selon l'algorithme montré à la figure 4. Préalablement à une étape O1, si aucun nombre aléatoire n'a été associé à une donnée, la machine virtuelle VM déclenche la génération d'un nombre aléatoire NAD dans le générateur 30 afin de l'écrire dans un registre RGD inclus dans l'espace de données de machine 51 de la mémoire EEPROM 5. Plus généralement selon cette réalisation, chaque fois qu'une donnée est écrite dans l'espace de mémoire RAM 61 ou 62 par la machine virtuelle VM, le nombre aléatoire NAD sert à transformer cette donnée.

En revenant à l'étape O2, les paramètres associés à l'instruction "new" servent à réserver la place nécessaire dans la mémoire 5 à l'objet OB créé par la machine virtuelle VM et à trouver toutes les informations de l'objet à créer. Les paramètres représentent un index qui permettent de retrouver des informations dans une table contenue dans le champ "constant_pool" contenu dans la machine virtuelle. Le constructeur d'objet dans la machine virtuelle retourne une référence REF qui fait office d'adresse du descripteur de l'objet créé.

Selon l'invention, la référence REF est transformée en une référence REFT qui est écrite sur le dessus de la pile de l'espace d'opérande 62 dans la mémoire RAM 6. Ainsi à l'étape suivante O3, la machine virtuelle VM applique la référence REF de l'objet créé et le nombre aléatoire NAD associé aux données à la fonction de transformation FT dans le transformateur 42 afin de produire la référence transformée REFT = FT(REF, NAD).

A titre d'exemple, les étapes suivantes O4, O5 et O6 concernent des manipulations de la référence transformée REFT, et non de la référence REF, dans la mémoire RAM 6 par la machine virtuelle MV. A l'étape O4, l'instruction suivante "dup" duplique la référence transformée REFT dans l'espace de mémoire 62 en ajoutant une copie de la référence REFT sur la pile d'opérandes. Cette duplication est relative à la création d'un autre objet identique à l'objet OB qui vient d'être créé dans la même classe. A l'étape suivante O5, l'instruction "invokespecial" ayant deux paramètres comme montré à la figure 3 appelle le constructeur de l'objet créé OB afin d'associer la référence transformée REFT à l'objet créé en interne dans la machine virtuelle VM. Puis l'instruction "astore" à l'étape O6 transfère la référence transformée REFT restante de l'espace de mémoire d'opérande 62 dans l'espace de mémoire de variable locale 61 dans la mémoire RAM 6 afin de manipuler l'objet, par exemple pour appeler une méthode applicable à cet objet.

Ainsi selon l'invention, toute référence REF à un objet est mémorisée sous sa forme transformée REFT dépendant du nombre aléatoire NAD dans la mémoire RAM 6. Plus généralement, cette transformation est applicable à tous les champs de l'objet et à toutes les données.

En variante, le générateur de nombres aléatoires 30 génère des nombres aléatoires respectivement associés à des types primitifs de données. Par exemple, lors de la création d'un objet, la machine virtuelle VM applique l'une ou plusieurs des transformations suivantes à des données INT de type entier, CHAR de type caractère, BOOL de type booléen (vrai/faux : "true/false"), REF du type référence et

D pour tous les autres types de données (float, double, etc...) :

INTT = FT (INT,NAI),

CHART = FT (CHAR,NAC),

BOOLT = FT (BOOL,NAB),

REFT = FT (REF,NAR),

DT = FT (D,NAD).

Dans ces relations, les données transformées INTT, CHART, BOOLT, REFT et DT résultent respectivement de transformations FT dans le transformateur 42 par l'application de la donnée initiale correspondante et d'un nombre aléatoire respectif NAI, NAC, NAB, NAR et NAD. Ainsi chaque fois qu'une application est instanciée, chaque donnée primitive INT, CHAR, BOOL, REF, D et le nombre aléatoire associé NAI, NAC, NAB, NAR, NAD sont appliqués par le transformateur 42 à la fonction de transformation FT afin d'écrire une donnée transformée INTT, CHART, BOOLT, REFT, DT dans la mémoire RAM 6.

Ceci contribue à remédier à des attaques qui tentent de faire exécuter des opérations indésirables sur des données de types primitifs différents ; l'attaquant ne peut pas prédire comment une donnée va être enregistrée ce qui l'empêche de réaliser son attaque. Par exemple, l'addition de deux références non transformées dans l'espace 62 selon la technique antérieure permet d'accéder à une donnée notamment sensible. Selon l'invention, l'addition de deux références transformées donne un résultat qui est complètement différent de l'addition des deux références initiales non transformées et qui a priori est totalement aléatoire. Comme les instructions sont également typées dans la machine virtuelle, une tentative d'addition par exemple du type entier de deux références transformées REFT accentuera encore

la différence entre le résultat obtenu et la somme des deux références initiales par l'opérateur addition de type référence.

On se réfèrera ci-après pour l'exécution d'une méthode de l'application téléchargée AP par exemple à des données INT de type entier associées au nombre aléatoire NAI écrit initialement dans le registre RGI de l'espace 51, tout en sachant que les données manipulées peuvent être n'importe quelles données de type primitif CHAR, BOOL, REF et D et le nombre aléatoire NAI peut être le nombre aléatoire respectif NAC, NAB, NAR et NAD.

En référence maintenant aux figures 5 et 6, l'exécution de l'application téléchargée AP dans la carte à puce C1 est décrite par exemple pour une partie du composant applicatif CP1 relative à une méthode comprenant trois instructions dans le composant CP1 dont les octets ont été transformés en fonction du nombre aléatoire NA1 à l'étape C5 (figure 2). Selon la figure 5, la méthode de l'application stockée dans l'espace de mémoire 51 comprend six octets transformés OCT. Les deux premiers octets transformés contiennent le code opération transformé et un paramètre d'une instruction transformée correspondant à l'instruction de chargement "iload" de type entier relative à une première variable locale VL1 enregistrée sous la forme transformée VLT1 = FT(VL1,NAI) dans l'espace de mémoire de variable locale 61. De même les troisième et quatrième octets transformés OCT dans la méthode montrée à la figure 5 correspondent à une instruction de chargement "iload"2 de type entier relative à une deuxième variable VL2 enregistrée sous la forme transformée VLT2=FT(VL2,NAI) dans l'espace de mémoire 61. Le

cinquième octet de la méthode est un octet transformé OCT d'une instruction "iadd" désignant une addition des deux variables VL1 et VL2 de type entier.

Comme montré à la figure 6, l'exécution de la méthode d'addition à cinq octets transformés OCT montrée à la figure 5, comprend essentiellement des étapes E1 à E10.

Tout d'abord pour exécuter les deux premières instructions de chargement, la machine virtuelle VM lit le nombre aléatoire NA1 dans le registre RG2 de l'espace de mémoire 60, à l'étape E1, le nombre aléatoire NA1 ayant été produit initialement par le générateur 30 lors du chargement de l'application AP.

A l'étape E2, chacun des octets transformés de l'instruction "iload" 1 est appliqué avec le nombre aléatoire NA1 à la fonction réciproque FT^{-1} dans le transformateur 43 qui produit l'instruction initiale "iload 1". La machine virtuelle VM exécute alors l'instruction "iload 1" en chargeant la variable transformée VLT1 = FT(VL1,NAI) au-dessus de la pile d'opérandes dans l'espace 62 de la mémoire RAM 6.

Puis l'instruction "iload 2" est traitée de la même manière aux étapes E4 et E5 que l'instruction précédente "iload 1" aux étapes E2 et E3 d'abord en récupérant l'instruction initiale "iload 2" dans le transformateur 43, puis en exécutant l'instruction "iload 2" afin de transférer la variable transformée VLT2 = FT(VL2,NAD) au-dessus de la variable transformée précédente VLT1 dans la pile de l'espace 62.

L'étape E6 est analogue aux étapes précédentes E2 et E4 afin de récupérer la troisième instruction iadd en appliquant le cinquième octet transformé dans la méthode montrée à la figure 5 et le nombre

aléatoire NA1 à la fonction réciproque FT dans le transformateur 43.

Puisque l'opération d'addition iadd ne peut être exécutée que sur les variables locales initiales, les deux variables locales à additionner VL1 et VL2 sont récupérées d'abord en lisant le nombre aléatoire NAI dans le registre RGI de l'espace de mémoire 51, ou en variante, le nombre aléatoire NAD associé à tous les types primitifs de données dans le registre RGD de l'espace de mémoire 51. La machine virtuelle lit ensuite également les variables locales transformées VLT2 et VLT1 au-dessus de la pile dans l'espace 62 et à l'étape E8 les applique chacune à la fonction réciproque FT^{-1} dans le transformateur 43 afin de récupérer les variables locales initiales VL1 et VL2 qui sont écrites dans deux registres de l'unité logique arithmétique dans la machine virtuelle VM. L'unité arithmétique exécute l'instruction "iadd" pour additionner les variables locales VL1 et VL2 en une somme SOM à l'étape E9. Le transformateur 42 transforme la somme SOM en l'appliquant avec le nombre aléatoire NAI, ou en variante NAD, à la fonction de transformation FT dans le transformateur 42 qui produit la somme transformée SOMT à l'étape E10. La somme transformée SOMT est finalement positionnée au-dessus de la pile dans l'espace de mémoire 62 de la mémoire RAM 6.

REVENDICATIONS

1 - Dispositif de traitement de données (1) comprenant un moyen de mémorisation (6) pour mémoriser au moins une application téléchargée (AP), initialement compilée en un langage intermédiaire, composée de plusieurs composants applicatifs (CP1) contenant chacun un identificateur (ID1) et des mots d'instruction (OC1), et un moyen d'exécution virtuel en langage intermédiaire (VM, 4, 5), caractérisé en ce qu'il comprend :

- un moyen générateur de nombres aléatoires (30) pour associer un nombre aléatoire (NA1) à un composant applicatif prédéterminé (CP1) de l'application téléchargée (AP),

- un premier moyen transformateur (42) inclus dans le moyen d'exécution virtuel pour appliquer chacun des mots d'instruction (OC1) dans le composant applicatif prédéterminé (CP1) et le nombre aléatoire associé (NA1) à une fonction de transformation (FT) afin de mémoriser des mots d'instruction transformés (OCT1) lors du téléchargement du composant applicatif prédéterminé, et

- un deuxième moyen transformateur (43) inclus dans le moyen d'exécution virtuel pour appliquer chacun des mots d'instruction transformés (OCT1) d'une partie du composant applicatif prédéterminé (CP1) et le nombre aléatoire associé (NA1) à la fonction réciproque (FT^{-1}) de la fonction de transformation (FT) afin de récupérer les mots d'instruction (OC1) composant ladite partie du composant applicatif prédéterminé pour exécuter ladite partie de composant ainsi récupérée.

2 - Dispositif conforme à la revendication 1, dans lequel les mots d'instruction (OC1) sont des octets de code opération (IN) et des octets de paramètre (PA).

3 - Dispositif conforme à la revendication 1, dans lequel chaque mot d'instruction a une longueur constante et correspond à une instruction complète.

4 - Dispositif conforme à l'une quelconque des revendications 1 à 3, comprenant un deuxième moyen de mémorisation (5) pour mémoriser l'application téléchargée (AP) mais comportant le composant applicatif prédéterminé avec des mots d'instruction transformés (OCT1).

5 - Dispositif conforme à l'une quelconque des revendications 1 à 4, dans lequel le moyen générateur (30) génère un nombre aléatoire (NAD) produit par le moyen d'exécution virtuel (VM), et le premier moyen transformateur (42) applique chaque donnée (REF) et le nombre aléatoire associé (NAD) à la fonction de transformation (FT) afin d'écrire une donnée transformée (REFT) dans le moyen de mémorisation (6) lorsque la donnée est produite par le moyen d'exécution virtuel (VM).

6 - Dispositif conforme à l'une quelconque des revendications 1 à 5, dans lequel le moyen générateur (30) génère des nombres aléatoires respectivement associés à des types primitifs de données, et le premier moyen transformateur (42) applique chaque donnée produite par le moyen d'exécution virtuel (VM) et le nombre aléatoire associé au type primitif de la donnée à la fonction de transformation (FT) afin

d'écrire l'autre donnée transformée dans le moyen de mémorisation (6) lorsque la donnée est produite par le moyen d'exécution virtuel (VM).

7 - Dispositif conforme à l'une quelconque des revendications 1 à 6, dans lequel le moyen générateur (30) associe des nombres aléatoires (NA1) respectivement à des identificateurs (ID1) de types de composant différents (CP1) afin que les mots d'instruction dans chaque composant soient appliqués à la fonction de transformation (FT) avec le nombre aléatoire associé au composant.

8 - Dispositif conforme à l'une quelconque des revendications 1 à 7, dans lequel la fonction de transformation (FT) et la fonction réciproque (FT^{-1}) sont des fonctions OU Exclusif.

9 - Dispositif conforme à l'une quelconque des revendications 1 à 8, dans lequel le moyen générateur de nombres aléatoires (30) est inclus dans ou en liaison avec un processeur (3) dans le dispositif.

10 - Dispositif conforme à l'une quelconque des revendications 1 à 8, dans lequel le moyen générateur de nombres aléatoires est inclus dans le moyen d'exécution virtuel (VM).

11 - Dispositif conforme à l'une quelconque des revendications 1 à 10, dans lequel le moyen de mémorisation est une mémoire à accès aléatoire (6) contenant des données échangées avec le monde extérieur au dispositif (1).

12 - Dispositif conforme à l'une quelconque des revendications 1 à 11, du type objet électronique portable tel que carte à puce (1).

1/4

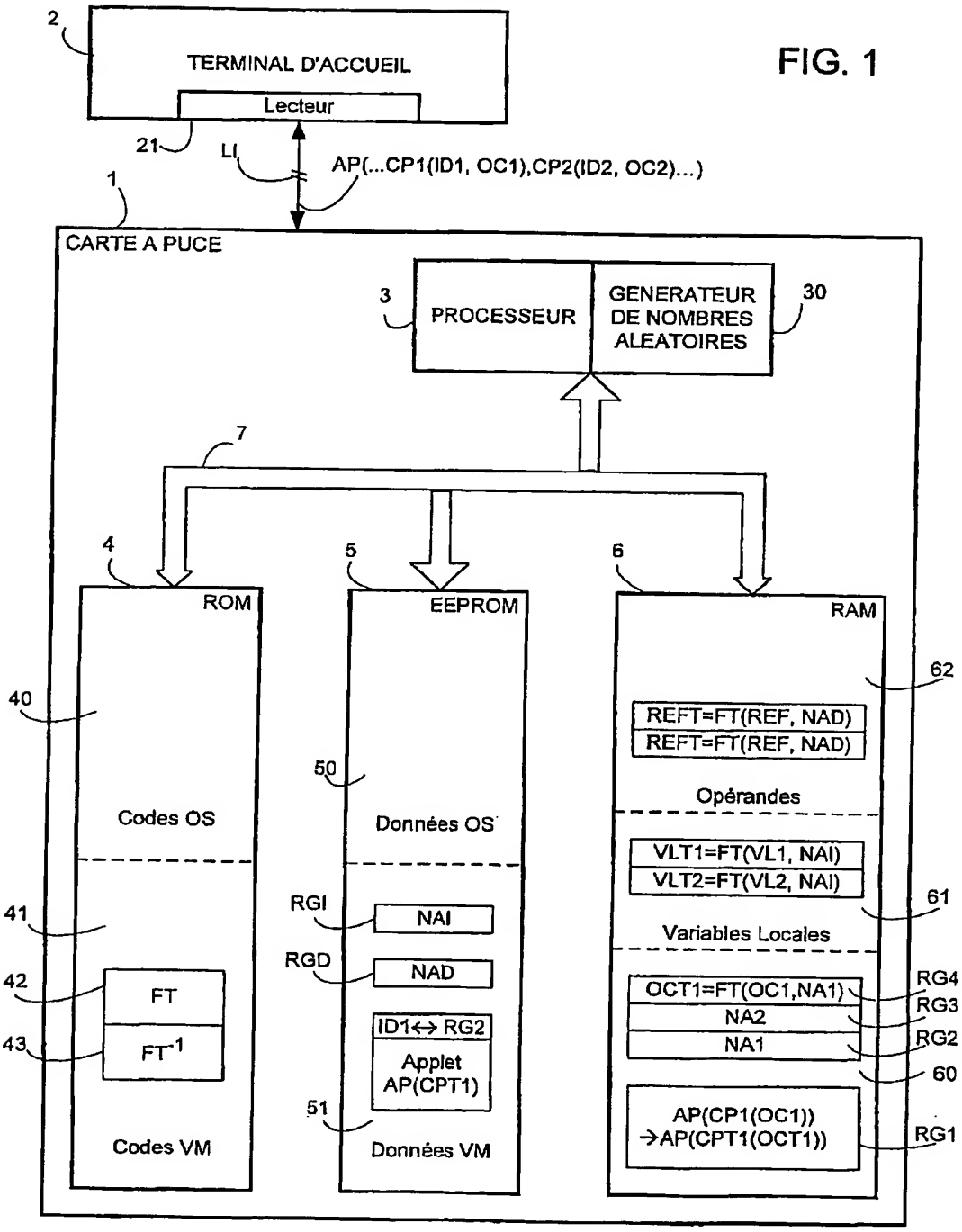

2
TERMINAL D'ACCUEIL
Lecteur
21
LI
AP(...CP1(ID1, OC1),CP2(ID2, OC2)...)
1
CARTE A PUCE
3
PROCESSEUR
GENERATEUR DE NOMBRES ALEATOIRES
30
7
4
ROM
5
EEPROM
6
RAM
40
Codes OS
41
42
FT
43
FT-1
Codes VM
50
Données OS
RGI
NAI
RGD
NAD
ID1↔ RG2
Applet AP(CPT1)
51
Données VM
62
REFT=FT(REF, NAD)
REFT=FT(REF, NAD)
Opérandes
VLT1=FT(VL1, NAI)
VLT2=FT(VL2, NAI)
61
Variables Locales
OCT1=FT(OC1,NA1)
NA2
NA1
RG4
RG3
RG2
60
AP(CP1(OC1))
→AP(CPT1(OCT1))
RG1

FIG. 1

FIG. 2

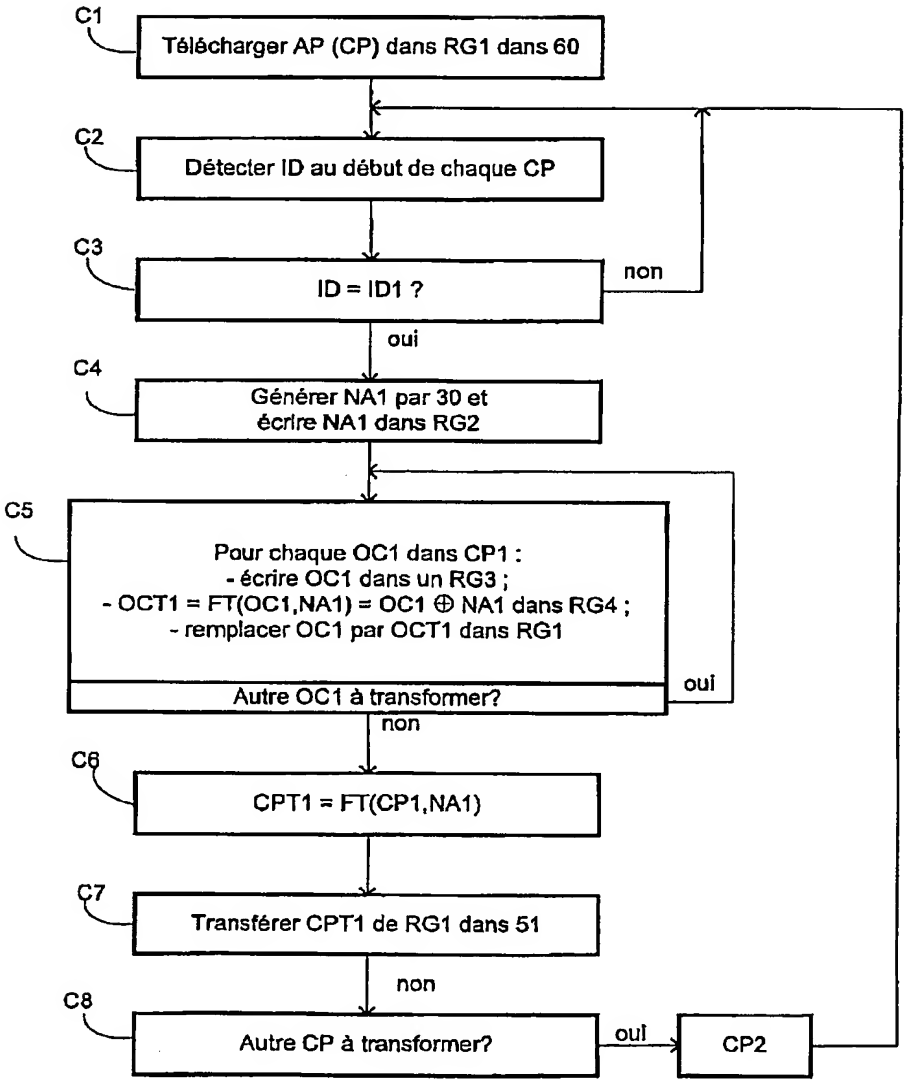

C1
Télécharger AP (CP) dans RG1 dans 60
C2
Détecter ID au début de chaque CP
C3
ID = ID1 ?
non
oui
C4
Générer NA1 par 30 et
écrire NA1 dans RG2
C5
Pour chaque OC1 dans CP1 :
- écrire OC1 dans un RG3 ;
- OCT1 = FT(OC1,NA1) = OC1 ⊕ NA1 dans RG4 ;
- remplacer OC1 par OCT1 dans RG1
Autre OC1 à transformer?
oui
non
C6
CPT1 = FT(CP1,NA1)
C7
Transférer CPT1 de RG1 dans 51
non
C8
Autre CP à transformer?
oui
CP2

FIG. 3

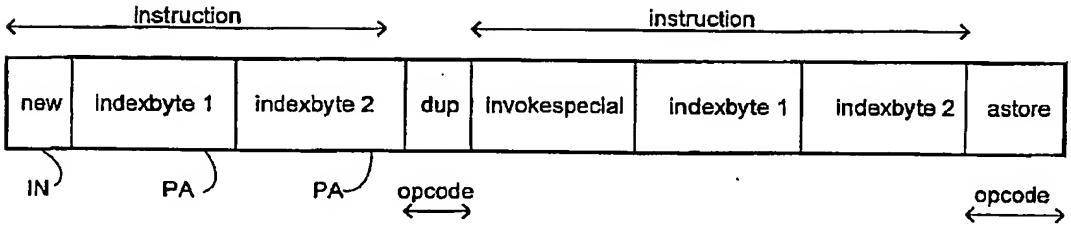

Instruction
instruction
new
indexbyte 1
indexbyte 2
dup
invokespecial
indexbyte 1
indexbyte 2
astore
IN
PA
PA
opcode
opcode

FIG. 4

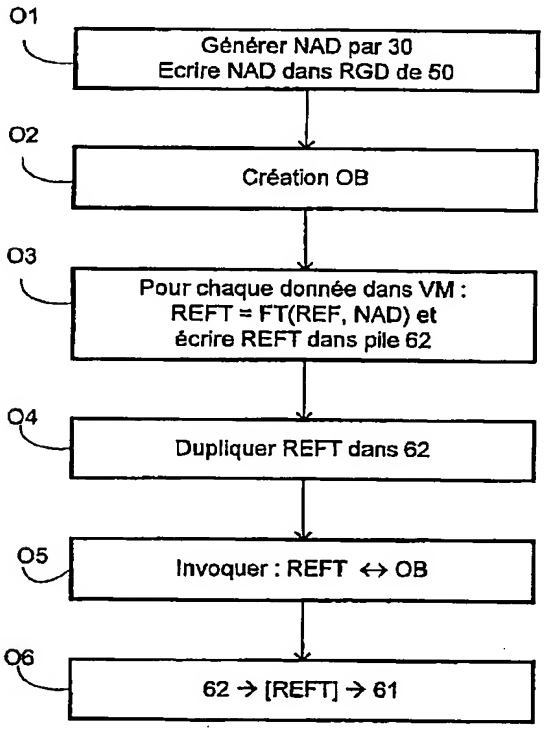

O1
Générer NAD par 30
Ecrire NAD dans RGD de 50
O2
Création OB
O3
Pour chaque donnée dans VM :
REFT = FT(REF, NAD) et
écrire REFT dans pile 62
O4
Dupliquer REFT dans 62
O5
Invoquer : REFT ↔ OB
O6
62 → [REFT] → 61

FIG. 5

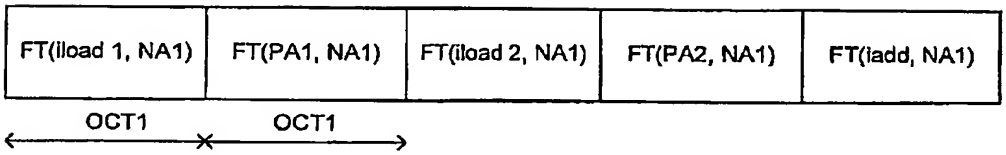
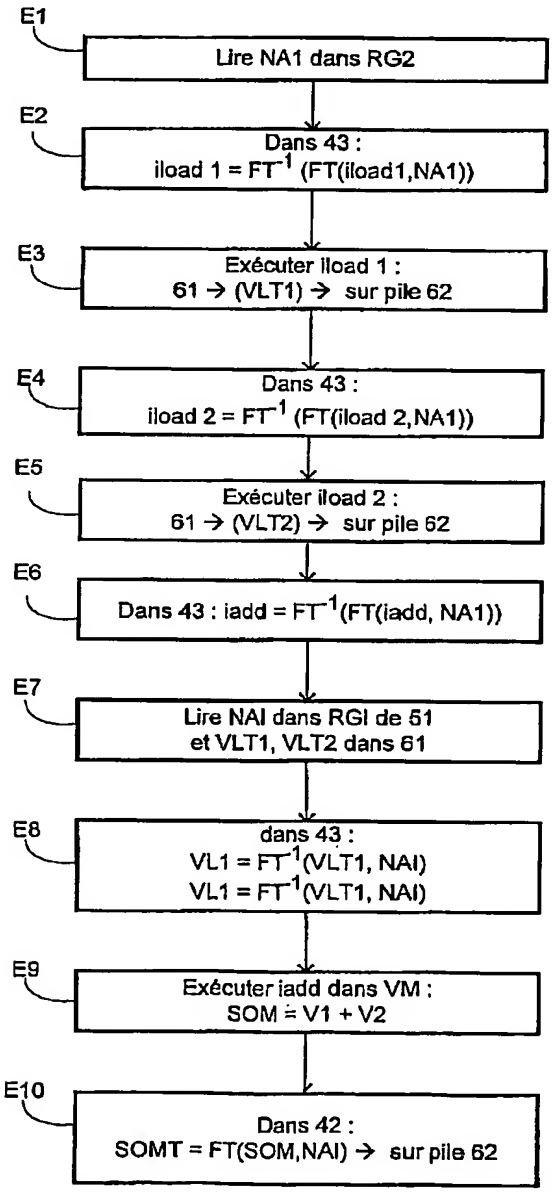

FT(iload 1, NA1)
FT(PA1, NA1)
FT(iload 2, NA1)
FT(PA2, NA1)
FT(iadd, NA1)
OCT1
OCT1

FIG. 6


E1
Lire NA1 dans RG2
E2
Dans 43 :
iload 1 = FT-1 (FT(iload1,NA1))
E3
Exécuter iload 1 :
61 → (VLT1) → sur pile 62
E4
Dans 43 :
iload 2 = FT-1 (FT(iload 2,NA1))
E5
Exécuter iload 2 :
61 → (VLT2) → sur pile 62
E6
Dans 43 : iadd = FT-1(FT(iadd, NA1))
E7
Lire NAI dans RGI de 51
et VLT1, VLT2 dans 61
E8
dans 43 :
VL1 = FT-1(VLT1, NAI)
VL1 = FT-1(VLT1, NAI)
E9
Exécuter iadd dans VM :
SOM = V1 + V2
E10
Dans 42 :
SOMT = FT(SOM,NAI) → sur pile 62